FIG. 12

```
typedef struct EncoderObj {
    IALG_Obj ialgObj;          /* IALG object MUST be first field */
    Int *workBuf;              /* pointer to on-chip scratch memory */
    Int *historyBuf;           /* previous frame's data in ext mem */
    ... ;
} EncoderObj ;

  Void algActivate (IALG_Handle handle)
  {
     EncoderObj *inst = (EncoderObj *)handle;

     /* copy history to beginning of on-chip working buf */
     memcpy(inst->workingBuf, inst->histBuf, HISTSIZE) ;
  }                                                                  } 1201
  Void encode (IALG_Handle handle,
                    Void *in[] , Void *out[] )
  {
     EncoderObj *inst = (EncoderObj *) handle;

     /* append input buffer to history in on-chip workBuf */
     memcpy (inst->workBuf + HISTSIZE, in HISTSIZE) ;

     /* encode data */
     ...
     /* move history to beginning of workbuf for next frame */
     memcpy (inst->workBuf, inst->workingBuf + FRAMESIZE, HISTSIZE) ;
  }

  Void algDeactivate (IALG_Handle handle)
  {
     EncoderObj *inst = (EncoderObj *) handle;                       } 1202

     /* save beginning of on-chip workBuf to history */
     memcpy (inst->histBuf, inst->workingBuf, HISTSIZE) ;
```

FIG. 13

```
typedef struct EncoderObj   {
      IALG_Obj    ialgObj                          /* IALG object MUST be first field */
      Int         *workBuf;                        /* pointer to on-chip scratch memory */
      Int         workBufLen;                      /* expressed in words per frame */
      ... ;
} EncoderObj;

typedef struct EncoderParams {
      Int frameDuration;                           /* expressed in ms per frame */
} ;
EncoderParams ENCODERATTRS = {5};                  /* default parameters */

Int algAlloc(IALG_Params *algParams, IALG_Fxns **p, IALG_MemRec memTab[] )
{
      EncoderParams *params = (EncoderParams *)algParams;
      if (params == NULL) {
          params = &ENCODERATTRS;                  /* use default parameters */
      }
      memTab [0] .size = sizeof (EncoderObj) ;
      memTab [0] .alignment = 1;
      memTab [0] .type = IALG_PERSIST;   ~1301
      memTab [0] .space = IALG_EXTERNAL; ~1302

      memTab [1] .size = params->frameDuration * 8 * sizeof(int) ;
      memTab [1] .alignment = 1;                   /* no alignment */
      memTab [1] .type = IALG_PERSIST;   ~1303
      memTab [1] .space = IALG_DARAM;    ~1304     /* dual-access on-chip */

      return (2);
}
```

```
typedef struct EncoderStatus {
   Bool voicePresent; /* voice in current frame? */
   ... ;
} EncoderStatus;

typedef enum {EncoderGetStatus, ...} EncoderCmd;

Void algControl (IALG_Handle handle,
                 IALG_Cmd cmd, IALG_Status *status)
{
   EncoderStatus *sptr = (EncoderStatus *)status;

   switch ((EncoderCmd)cmd) {
      case EncoderGetStatus:
         sptr->voicePresent = ...;
         ...
      case EncoderSetMIPS:
         ...
   }
}
```

FIG. 14

FIG. 15

```
typedef struct EncoderObj {
      IALG_Obj    ialgObj                    /* IALG object MUST be first field */
      Int         *workBuf;
      Int         workBufLen;
      ... ;
} EncoderObj;
Int algFree(IALG_Handle handle, IALG_MemRec memTab[] )
{
      EncoderObj *inst = (EncoderObj *)handle;

      algAlloc(NULL, memTab) ;                /* get default values first */

      memTab[1] .size = inst->workBufLen * sizeof(Int) ;
      memTab[1] .base = (Void *)inst->workBuf;
      return(2);
}

Int algAlloc(IALG_Params *params, IALG_MemRec memTab[] )
{
      memTab[0] .size = sizeof (EncoderObj) ;
      memTab[0] .alignment = 1;
      memTab[0] .type = IALG_PERSIST;
      memTab[0] .space = IALG_EXTERNAL;

      memTab[1] .size = 80;                   /* 10ms @ 8KHz */
      memTab[1] .alignment = 1;               /* no alignment */
      memTab[1] .type = IALG_PERSIST;
      memTab[1] .space = IALG_DARAM;          /* dual-access on-chip */

      return (2) ;
}
```

FIG. 16

```
typedef struct EncoderObj {
      IALG_Obj    ialgObj                    /* IALG object MUST be first field */
      Int         workBuf;                   /* pointer to on-chip scratch memory */
      Int         workBufLen;                /* workBuf length (in words) */
      ... ;
} EncoderObj ;

Int algInit(IALG_Handle handle,
          IALG_MemRec memTab[], IALG_Handle p, IALG_Params *algParams)
{
      EncoderObj *inst = (EncoderObj *)handle;
      EncoderParams *params = (EncoderParams *)algParams;

      if (params == NULL) {
          params = &ENCODERATTRS;            /* use default parameters */

      }

      inst->workBuf = memTab[1] .base;
      inst->workBufLen = params->frameDuration * 8;
      ...

      return (IALG_EOK) ;
}
```

FIG. 17

```
typedef struct EncoderObj {
   IALG_Obj ialgObj  /* IALG object MUST be first field */
   int workBuf;      /* pointer to on-chip scratch memory */
   Int workBufLen; /* workBuf length (in words) */
   ... ;
} EncoderObj ;

algMoved(IALG_Handle handle,
              IALG_Params *algParams, IALG_MemRec memTab[] )
{
1701  EncoderObj *inst = (EncoderObj *)handle;

   inst->workBuf = memTab[1] .base;
}
```

FIG. 18

```
#define NUMBUF  3          /* max number of my memory requests */
extern IALG_Fxns *subAlg;  /* sub-algorithm used by this alg */
Int algNumAlloc(Void)
{
    return (NUMBUF + subAlg->algNumAlloc()) ;
}

Int algAlloc(const IALG_Params *p, struct IALG_Fxns **pFxns,
             IALG_MemRec memTab)
{
    Int n;

    /* initialize my memory requests */
    ...
    /* initialize sub-algorithm's requests */
    n = subAlg->algAlloc(..., memTab) ;
    return (n + NUMBUF) ;
}
```

1801

FIG. 19

```
/*----------------------*/
/*      TYPES AND CONSTANTS      */
/*----------------------*/
#define IRTC_ENTER          0
#define IRTC_CLASS1         1
#define IRTC_CLASS2         2
#define IRTC_CLASS3         3
#define IRTC_CLASS4         4
#define IRTC_CLASS5         5
#define IRTC_CLASS6         6
#define IRTC_CLASS7         7

/*
 *  ======== IRTC_Handle ========
 *  Handle to module's trace instance object
 */
typedef struct IRTC_Obj *IRTC_Handle;

/*
 * ========IRTC_Mask ========
 */
typedef LgUns IRTC_Mask ;

/*
 * ========IRTC_Fxns ========
 */
typedef struct IRTC_Fxns {
    Void        *implementationId;
    Void        (*rtcBind) (LOG_Obj *log) ;
    IRTC_Mask   (*rtcGet) (IRTC_Handle) ;
    Void        (*rtcSet) (IRTC_Handle, IRTC_Mask mask) ;
} IRTC_Fxns;
```

1901

FIG. 20A

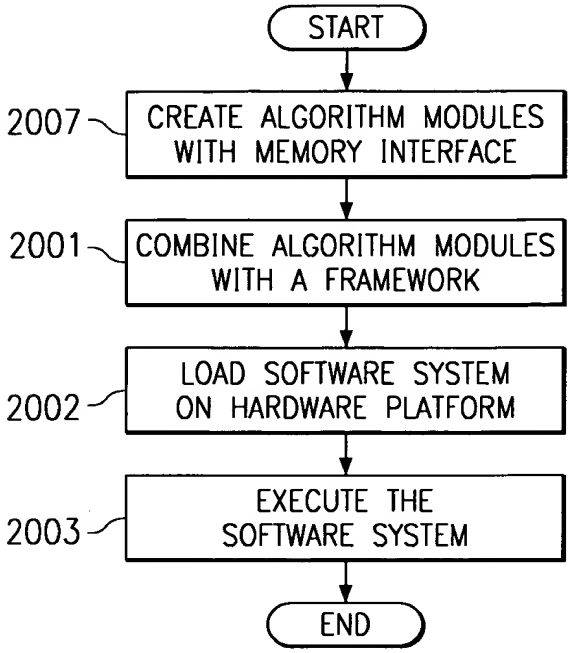
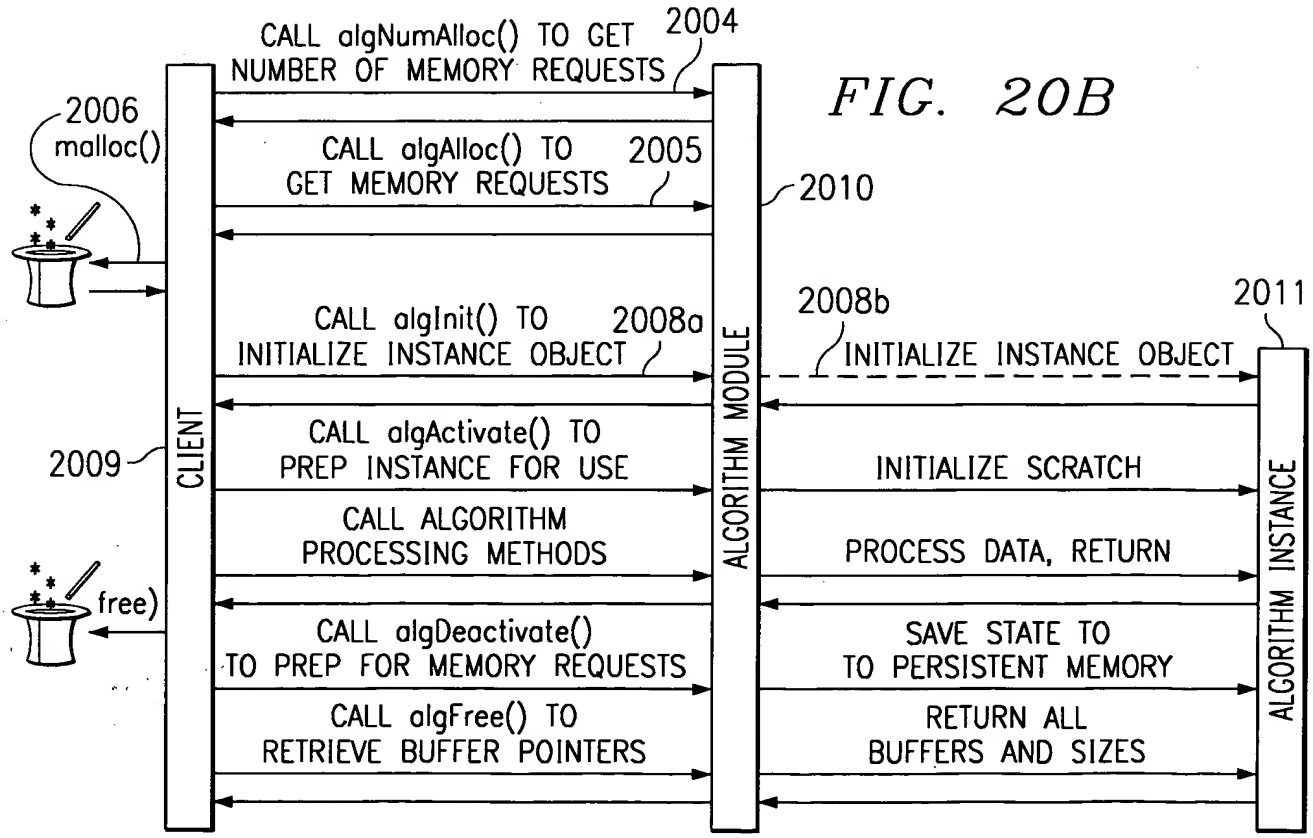


FIG. 20B



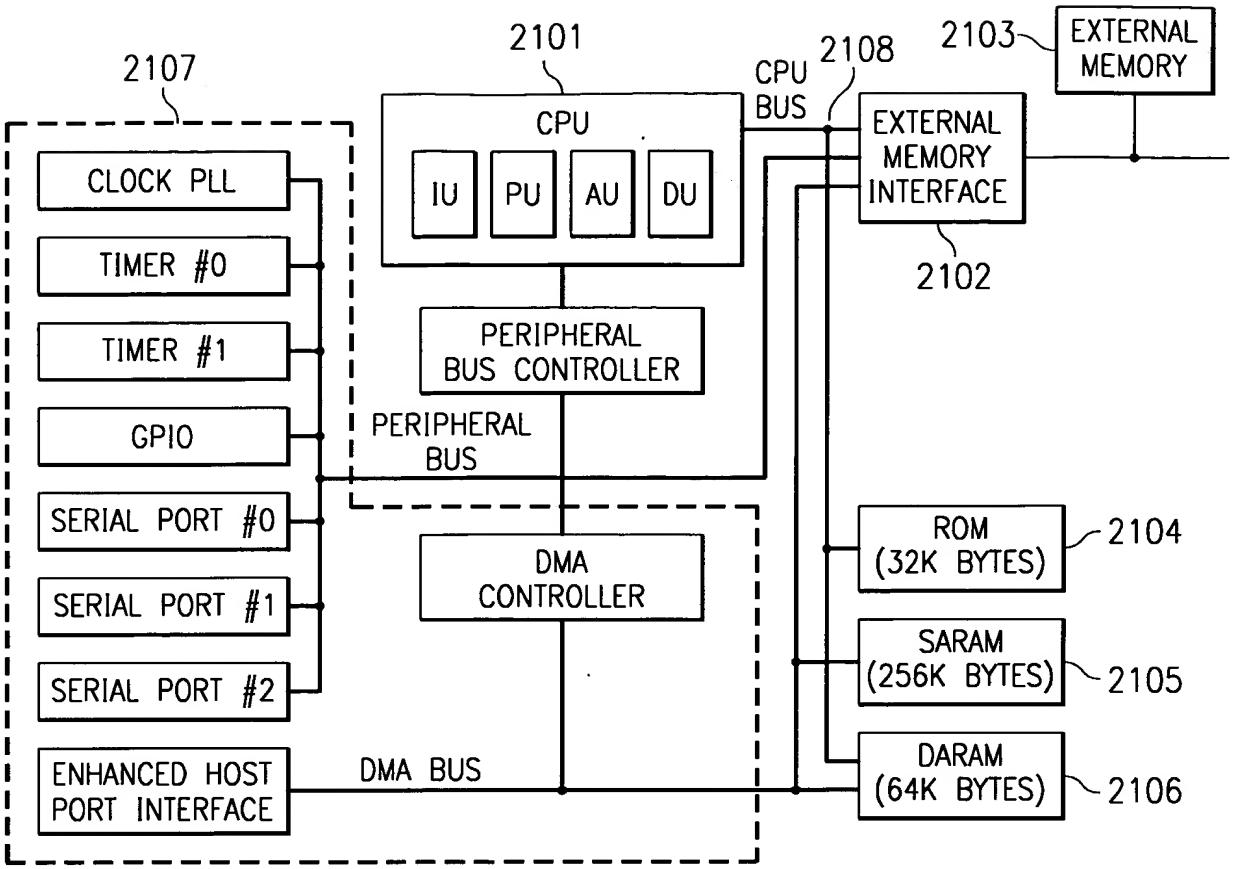


FIG. 21

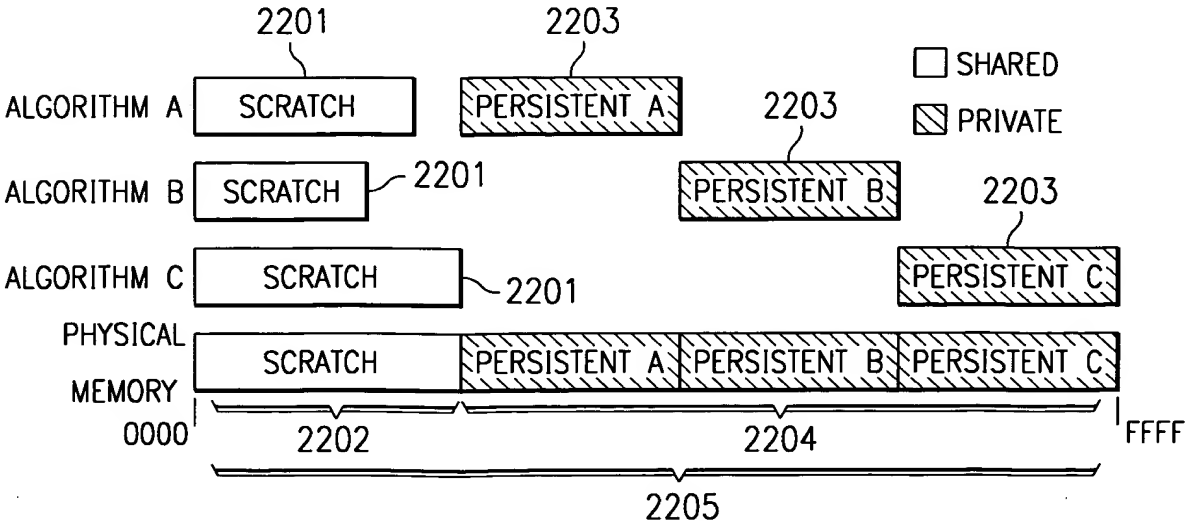


FIG. 22

FIG. 23A

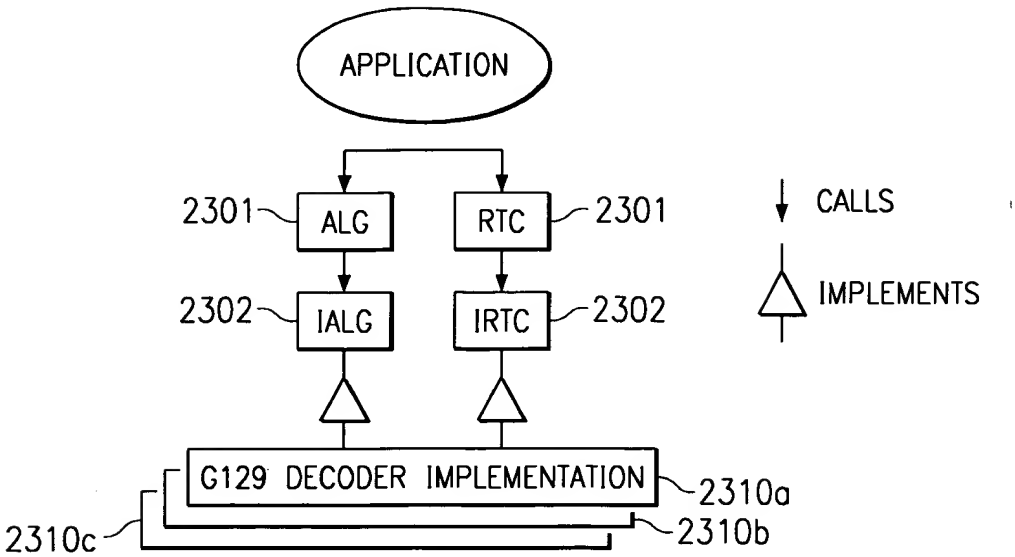


FIG. 23B

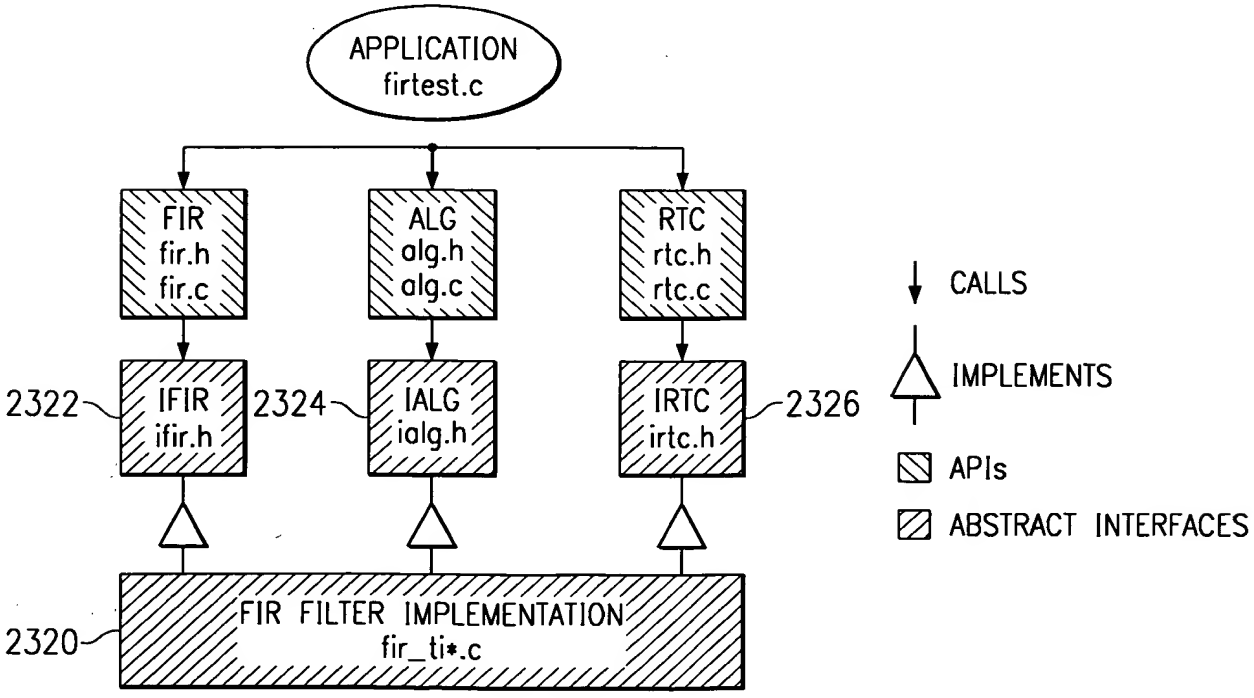


FIG. 24

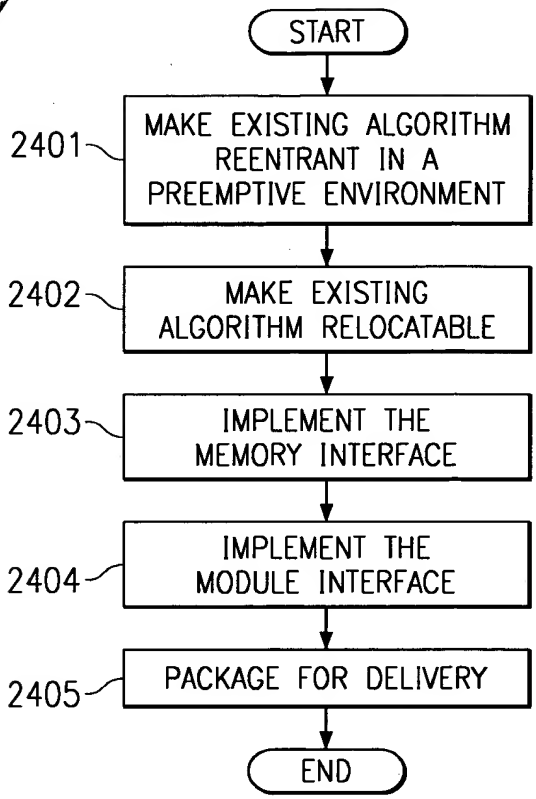


FIG. 26A

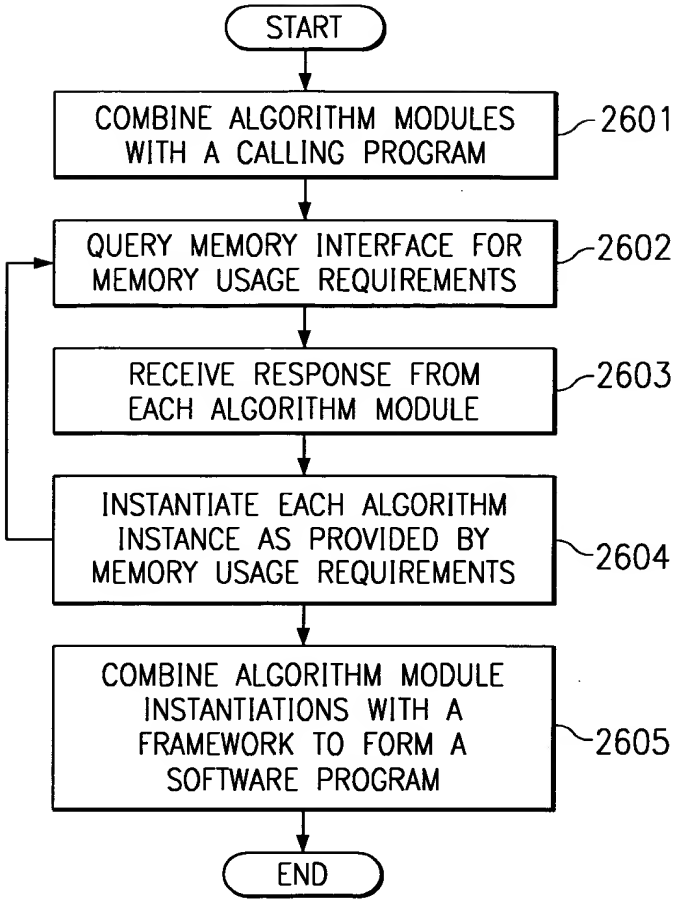


FIG. 25

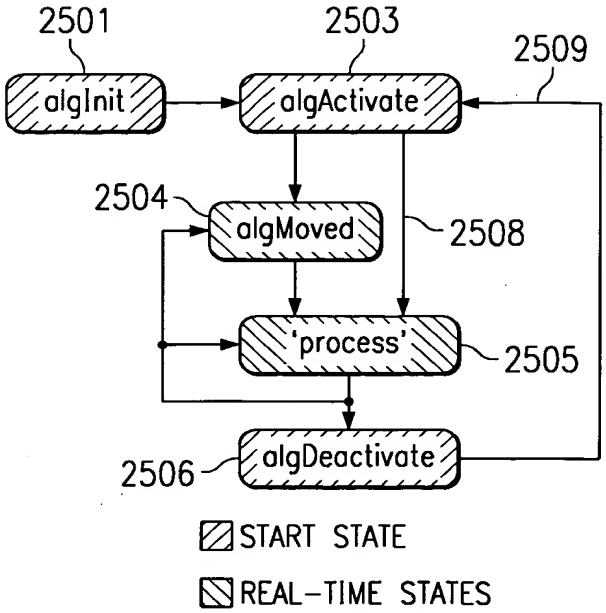


FIG. 27

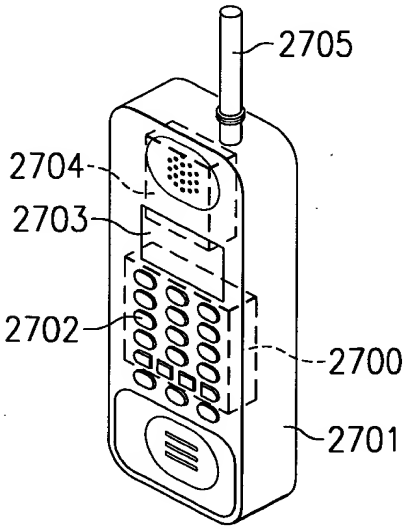
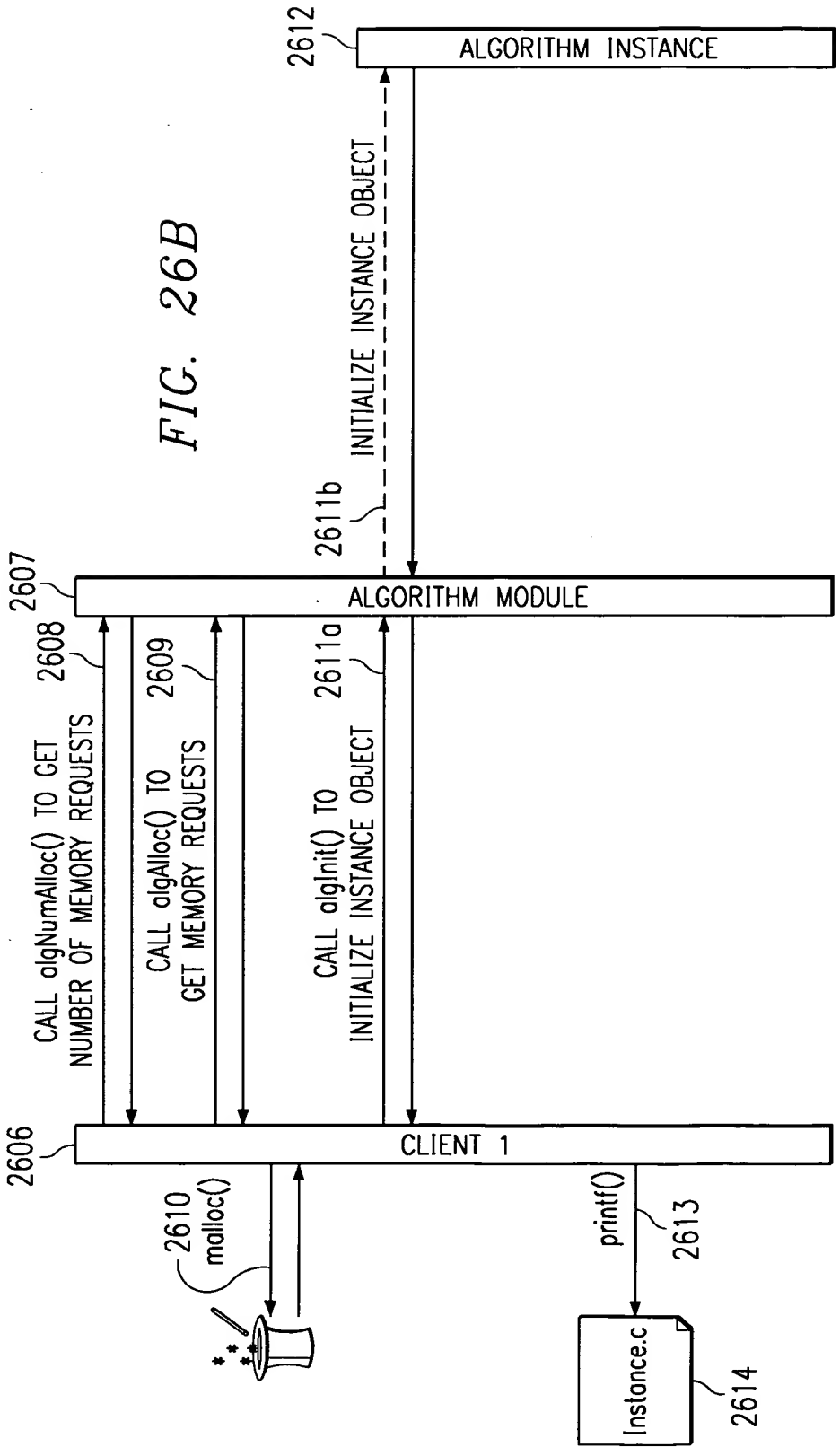


FIG. 26B



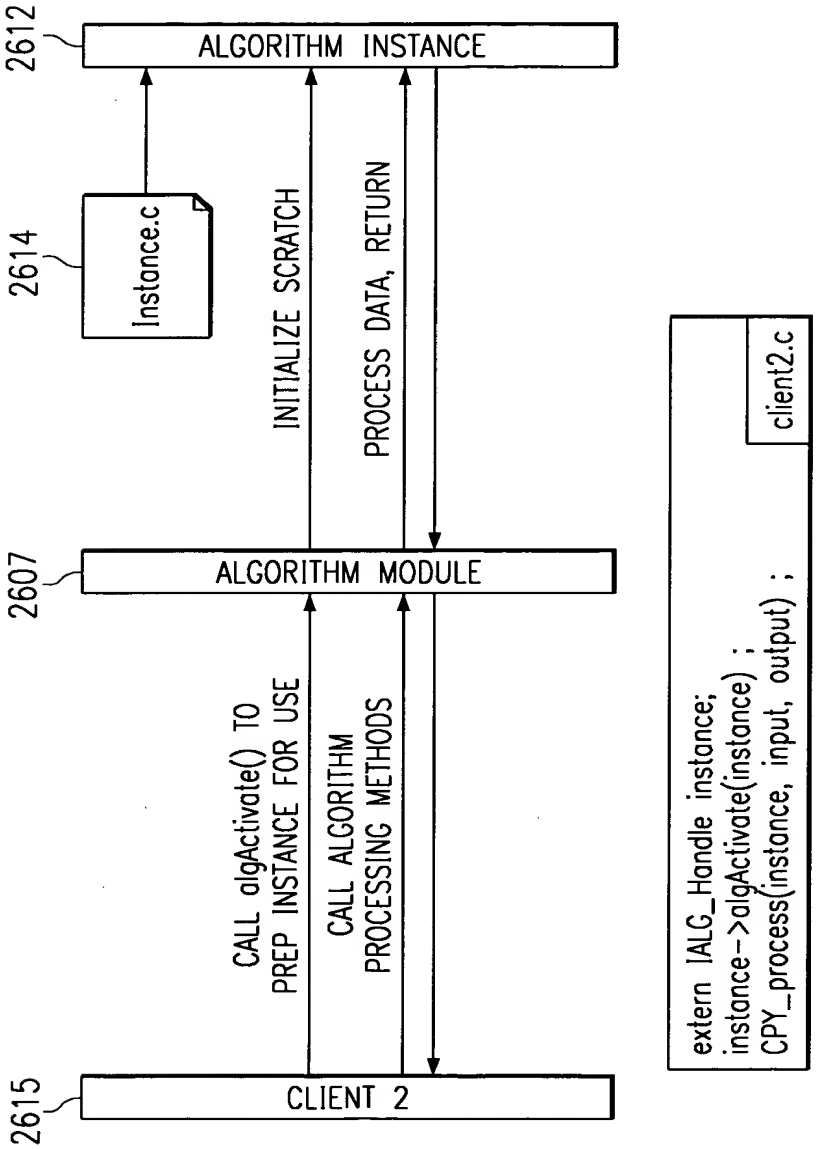


FIG. 26C

FIG. 1A

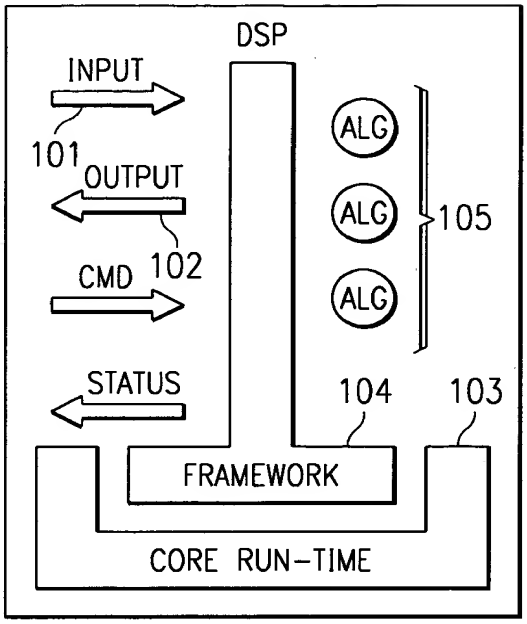


FIG. 1B

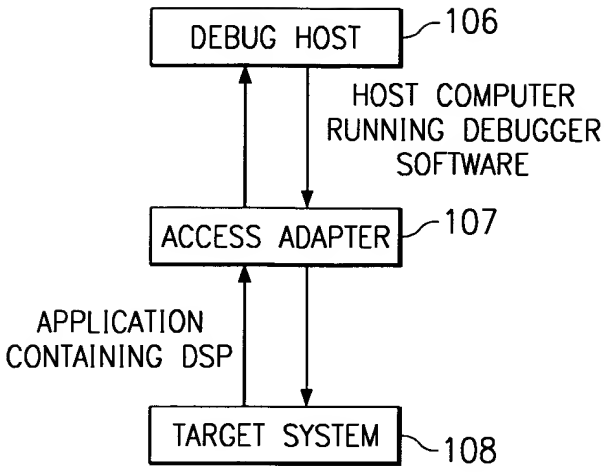
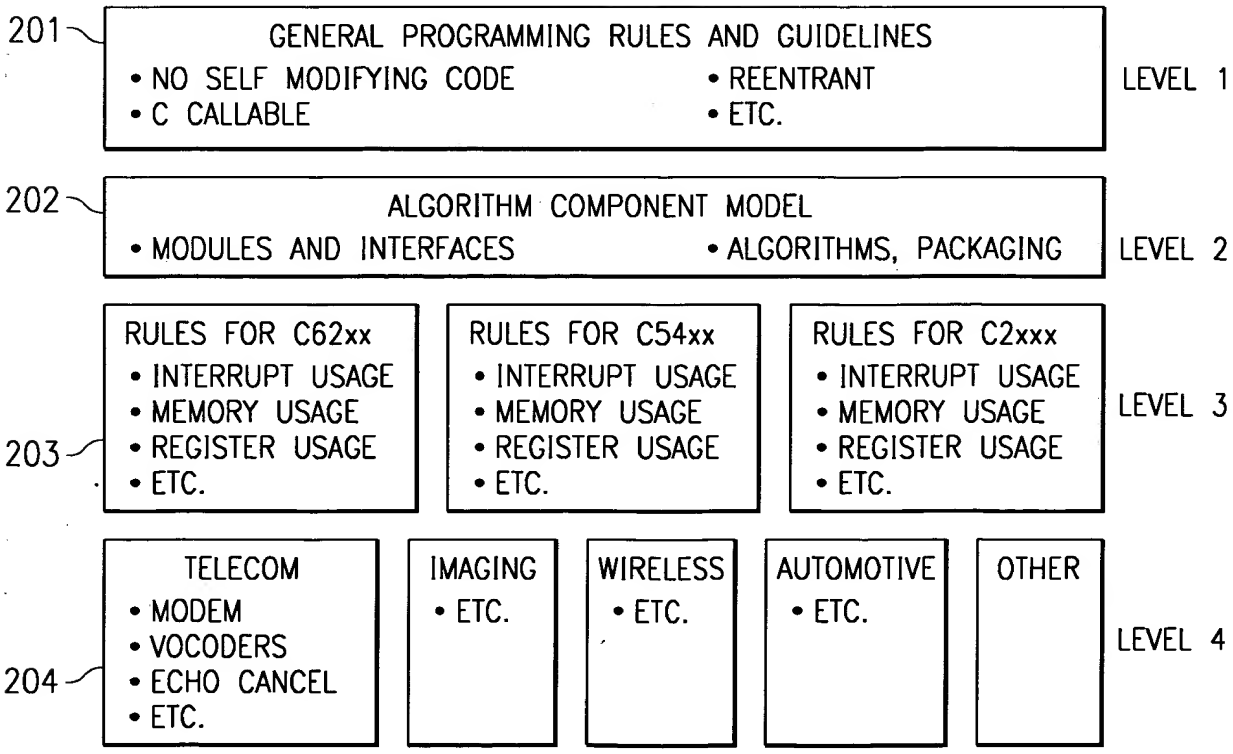


FIG. 2



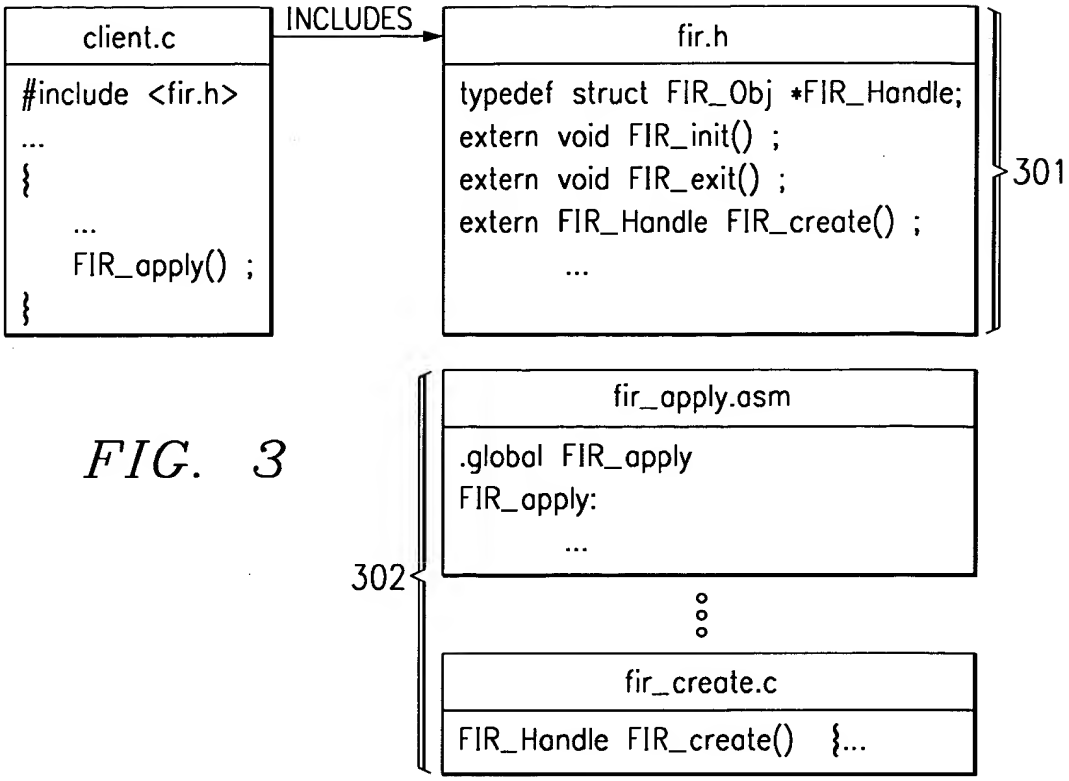


FIG. 3

FIG. 4

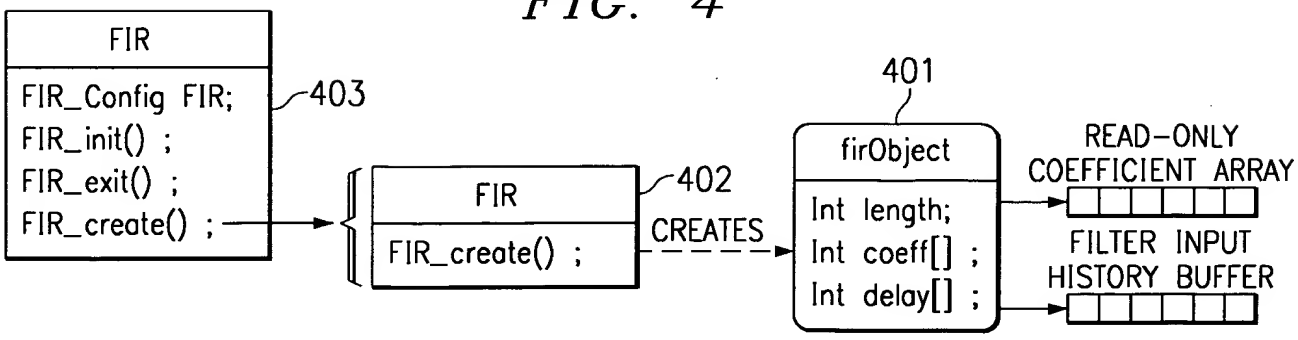
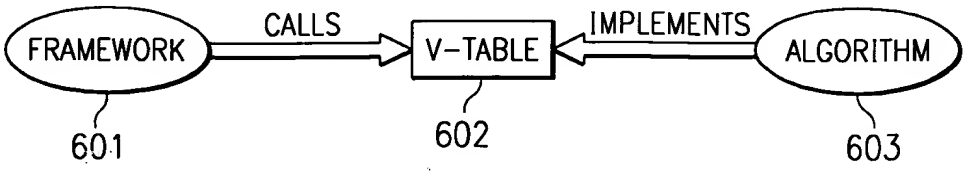


FIG. 6



```
501   void FIR_init (void)
      {
      }
      void FIR_exit (void)
      {
      }
```

FIG. 5

```
      typedef FIR_Params  {  /* FIR_Obj creation parameters */
          int frameLen ;     /* input/output frame length */
          int *coeff ;       /* pointer to filter coefficients */
      }  FIR_Params ;

505   FIR_Params FIR_PARAMS = { 64, NULL } ; /* default parameters */

      typedef struct FIR_Obj { /* FIR_Obj definition */
          int hist [16] ;    /* previous input value */
          int frameLen ;     /* input frame length */
          int *coeff ;
      } FIR_Obj ;
```

```
      FIR_Handle FIR_create (FIR_Obj *fir, const FIR_Params *params)
      {
          if (fir != NULL) {
              if (params == NULL)  { /* use defaults if params is NULL */
                  params = &FIR_PARAMS ;
              }
502           fir->frameLen = params->frameLen ;
              fir->coeff = params->coeff ;
              memset(fir->hist, 0, sizeof (fir->hist)) ;
          }

          return (fir) ;
      }
```

```
503   void FIR_delete (FIR_Handle fir)
      {
      }
```

```
      void FIR_apply (FIR_Handle fir, int in[] , int out[] )
      {
          int i ;

504       /* filter data using coefficients fir->coeff and
                history fir->hist */
          for (i = 0 ; i < fir->frameLen; i++)  {
                out [i] = filter(in[i], fir->coeff, fir->hist) ;
          }
      }
```


```
/*----------------------*/
/* TYPES AND CONSTANTS */
/*----------------------*/
#define IALG_DEFMEMRECS 4                /* default number of memory records */
#define IALG_OBJMEMREC 0                 /* memory record index of instance object */
#define IALG_SYSCMD 256                  /* minimum "system" IALG_Cmd value */
#define IALG_EOK 0                       /* successful return status code */
#define IALG_EFAIL -1                    /* unspecified error return status code */

typedef enum IALG_MemAttrs {
    IALG_SCRATCH,                        /* scratch memory */
    IALG_PERSIST,                        /* persistent memory */
    IALG_WRITEONCE                       /* write-once persistent memory */
} IALG_MemAttrs;

#define IALG_MPROG 0x0008                /* program memory space bit */
#define IALG_MXTRN 0x0010                /* external memory space bit */

/*
* ===== IALG_MemSpace =====
*/
typedef enum IALG_MemSpace {
    IALG_EPROG = IALG_MPROG | IALG_MXTRN,  /* external program memory */
    IALG_IPROG = IALG_MPROG,               /* internal program memory */
    IALG_ESDATA = IALG_MXTRN + 0,          /* off-chip data memory (accessed sequentially) */
    IALG_EXTERNAL = IALG_MXTRN + 1,        /* off-chip data memory (accessed randomly) */
    IALG_DARAM0 = 0,                       /* dual access on-chip data memory */
    IALG_DARAM1 = 1,                       /* dual access on-chip data memory */
    IALG_SARAM = 2,                        /* single access on-chip data memory */
    IALG_SARAM0 = 2,                       /* block 0, equivalent to IALG_SARAM */
    IALG_SARAM1 = 3                        /* block 1, if independant blocks required */
} IALG_MemSpace;

typedef struct IALG_MemRec {
    Int size;                              /* 'sizeof' memory request in MAUs (minimum address-able unit) */
    Int alignment;                         /* alignment requirement (in MAUs) */
    IALG_MemSpace space;                   /* allocation space */
    IALG_MemAttrs attrs;                   /* memory attributes */
    Void *base;                            /* base address of allocated buf */
} IALG_MemRec;
```

706 711 712 713 709 714 708 710

FIG. 7A

FIG. 7B

```
/*
 * ======== IALG_Obj ========
 * Algorithm instance object definition
 *
 * All XDAIS algorithm instance objects *must* have this structure as their first element.
 * However, they do not need to initialize it; initialization of this sub-structure is done by
 * the "framework".
 */

typedef struct IALG_Obj {
    struct IALG_Fxns *fxns;
} IALG_Obj;                                                    } 705

/*
 * ======== IALG_Handle ========
 * Handle to an algorithm instance object

 */
typedef struct IALG_Obj *IALG_Handle;                          } 707
/*

 * ======== IALG_Params ========
 * Algorithm instance creation parameters
 * All XDAIS algorithm parameter structures *must* have a this as their first element.
 */

typedef struct IALG_Params {
    Int size; /* number of MAUs (i.e. the 'sizeof') the structure */   } 703
} IALG_Params;

/*
 * ======== IALG_Status ========
 * Pointer to algorithm specific status structure
 * All XDAIS algorithm status structures *must* have this as their first element.
 */

typedef struct IALG_Status {
    Int size; /* number of MAUs (i.e. the 'sizeof') the structure */   } 704
} IALG_Status;

/*
 * ======== IALG_Cmd ========
 * Algorithm specific command. This command is used in conjunction with IALG_Status to
 * get and set algorithm specific attributes via the algControl method.
 */
typedef unsigned int IALG_Cmd;
```

FIG. 7C

```
typedef struct IALG_Fxns {
    Void        *implementationId;      /* 702 */
    Void        (*algActivate) (IALG_Handle) ;
    Int         (*algAlloc) (const IALG_Params *, struct IALG_Fxns **, IALG_MemRec *) ;
    Int         (*algControl) (IALG_Handle, IALG_Cmd, IALG_Status *) ;
    Void        (*algDeactivate) (IALG_Handle) ;
    Int         (*algFree) (IALG_Handle, IALG_MemRec *) ;
    Int         (*algInit) (IALG_Handle, const IALG_MemRec *, IALG_Handle, const
IALG_Params      *) ;
    Void        (*algMoved) (IALG_Handle, const IALG_MemRec *, IALG_Handle, const
IALG_Params      *) ;
    Int         (*algNumAlloc) (Void) ;
} IALG_Fxns;
```

701

FIG. 8

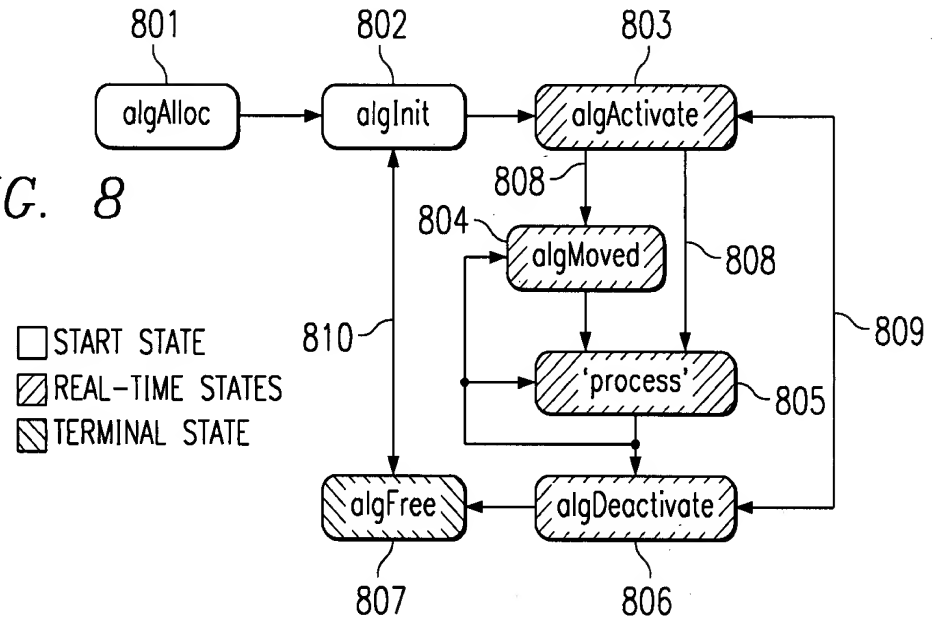


FIG. 9

```
client ()
{
      FIR_Params stdParams;
      FIR_TI_Params tiParams;

      stdParams = FIR_PARAMS;                  /* initialize all fields to defaults */
      stdParams.coeff = ...;                   /* initialize selected parameters */
      fxns->algAlloc(&stdParams, ...) ;        /* pass parameters to algorithm */

      tiParams = FIR_TI_PARAMS;                /* initialize all fields to defaults */
      tiParams.coeff = ...;                    /* initialize selected parameters */
      fxns->algAlloc(&tiParams, ...) ;         /* pass parameters to algorithm */
}
Int FIR_TI_algAlloc(IALG_Params *clientParams, ...)
{
      FIR_TI_Params params = FIR_TI_PARAMS;

      /* client passes in parameters, use them to override defaults */
      if (clientParams != NULL) {
      memcpy(&params, clientParams, clientParams->size) ;

      }

      /* use params as the complete set of parameters */
            :

}
```

FIG. 11

```
Void FIR_apply (FIR_Handle alg, Int *in[], Int *out[] )
{
   /* do app specific initialization of scratch memory */
   if (alg->fxns->ialg.algActivate != NULL) {
     alg->fxns->ialg_algActivate(alg) ;
   }
   /* filter data */
   alg->fxns->filter(alg, in, out) ;

   /* do app specific store of persistent data */
   if (alg->fxns->ialg.algDeactivate != NULL) {
     alg->fxns->ialg.algDeactivate(alg) ;
   }
}
```

FIG. 10

```
#define MAXMEMRECS  16

typedef struct ALG_Obj {
   IALG_Fxns    fxns;                    /* algorithm functions */
} ALG_Obj;

IALG_Handle ALG_create (IALG_Fxns *fxns, IALG_Params *params)
{
   IALG_MemRec   memTab [MAXMEMRECS] ;
   IALG_Handle   alg = NULL;
   Int           n;

   if (fxns->algNumAlloc () <= MAXMEMRECS) {
      n = fxns->algAlloc(params, memTab) ;
      if (allocMemory(memTab, n)) {
   alg = (IALG_Handle)memTab[0] .base;
1001 alg->fxns = fxns ;
   if (fxns->algInit(alg, memTab, params) != IALG_EOK) {
      fxns->algFree(alg, memTab) ;
      freeMemory(memTab, n) ;
      alg = NULL;
   }
      }
   }

   return (alg) ;
}

Void ALG_delete (IALG_Handle alg)
{
   IALG_MemRec memTab [MAXMEMRECS] ;
1002 Int n;

   n = alg->fxns->algFree(alg, memTab) ;
   freeMemory(memTab, n) ;
}
```